File No. 1410/7010-30
Form C28-0323-1

Systems Reference Library

IBM 1410/7010 Operating System (1410-PR-155) Random-Processing Scheduler—1410-IO-967

This publication is a reference text for use of the Random-Processing Scheduler, an optional component of the IBM 1410/7010 Operating System. Providing facilities for the efficient handling of input/output operations in random-processing applications, the Random-Processing Scheduler augments the Basic Input/Output Control System component of the Operating System. The manual discusses the random-processing concepts applicable to the Random-Processing Scheduler, and provides detailed information concerning the macro-instructions, Dcfine the File statements, and Define Area statements associated with the Random-Processing Scheduler.

This material is intended for use by programmers and systems analysts who have knowledge of the Basic Input/Output Control System and the IBM 1410/7010 Autocoder language. Information covering these two topics is presented in the following publications: *IBM 1410/7010 Operating System; Basic Input/Output Control System*, Form C28-0322. *IBM 1410/7010 Operating System; Autocoder*, Form C28-0326.

MAJOR REVISION (November, 1963)
This publication is a major revision of and supersedes the publication, *IBM 1410/7010 Operating System; Random-Processing Scheduler: Preliminary Specifications,* Form C28-0323, with its associated Technical Newsletter (N28-2003).

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D91, PO Box 390, Poughkeepsie, N. Y.

© 1963 by International Business Machines Corporation

Contents

Introduction

Purpose of This Publication

The purpose of this publication is to describe the capabilities of the Random-Processing Scheduler, and the function and use of the macro-instructions and DTF (Define the File) and DA (Define Area) statements needed to utilize the capabilities of the Scheduler in random processing applications involving the use of IBM 1301 Disk Storage.

Purpose of the Random-Processing Scheduler

The Random-Processing Scheduler is an optional component of the IBM 1410/7010 Operating System, providing facilities for efficient handling of input/output operations involving random access devices.

The goal of the Random-Processing Scheduler is to permit a random processing program to be planned as if using a serial "read a record, process it, write it" procedure. Actual execution of the program, however, is controlled by the Random-Processing Scheduler (hereafter called the Scheduler). The order in which records will be processed is determined by the Scheduler so that maximum operating efficiency is obtained.

The Random-Processing Scheduler is designed to be of maximum value for applications involving the updating of files resident in disk storage, where the data to be used in updating these files is not batched by type or sorted beforehand. The advantages of the Scheduler are lost if it is used in a sequential processing application.

By providing routines for the supervision and concurrent scheduling of input/output operations in a random processing application, the Scheduler relieves the user of the major programming task of designing an efficient random processing input/output control system.

Machine and Operating System Requirements

The machine requirements for assembling source programs written in the Autocoder language are specified in the publication, *IBM 1410/7010 Operating System; System Generation,* Form C28-0352. The machine requirements for execution of the object program depend upon the combined requisites of the System Monitor and the user's program.

The Random-Processing Scheduler requires of the Operating System that its Resident Monitor be conditioned to handle interrupt exits and 1301 Disk Storage. This is accomplished during generation of the Operating System by entries in the Resident IOCS Definitions macro-statements.

Prerequisites

It is assumed that the reader of this publication is familiar with fundamentals of programming the IBM 1410 or 7010 Data Processing System. To successfully prepare programs utilizing the Scheduler, the reader should also be familiar with the following publications:

IBM 1410/7010 Operating System; Basic Concepts, Form C28-0318
IBM 1410/7010 Operating System; System Monitor, Form C28-0319
IBM 1410/7010 Operating System; Autocoder, Form C28-0326
IBM 1410/7010 Operating System; Basic Input/Output Control System, Form C28-0322

Random Processing Concepts

Random access devices make it possible to directly pick out wanted data from a file, in a sequence established by the program using the data, not by the data arrangement. Also present is the capability of accessing data from a number of files in a sequence dictated by program need, not by file arrangement. In contrast, sequential access devices, such as magnetic tape, make data available only in a fixed, serial order.

Handling Disk Data

The arrangement of data and program requirements determine the method used in moving information to and from disk storage. Three methods of moving disk data are in general use and are known as *single-record, sequential,* and *random* processing.

Single-Record Processing

Single-record processing is defined as obtaining disk records of *any* format, brought from *any* area of disk storage, in *any* arbitrary order of addresses. It is the most flexible method of moving data to and from disk storage. The primary use of this method is in applications where the precise nature and the location of the data to be moved are not known at the time the program is written.

Sequential Processing

Sequential processing is defined as obtaining disk records in the order of ascending addresses. When information is transferred sequentially there is virtually no seek time involved if the records are arranged correctly on disk. Therefore, it is the fastest method of moving large amounts of data to and from disk storage. In many respects sequential processing is similar to tape processing, and, because of this, is not suitable for applications requiring arbitrary, or random, retrieval of disk data.

Random Processing

Random processing is defined as obtaining disk records of *uniform* format, belonging to specific files, in any arbitrary order of addresses. It is the most widely used method of reading and writing disk data and lends itself to any application where data in a specific file must be processed in an arbitrary order. The Random-Processing Scheduler is designed to implement this type of operation. The next section expands on this concept.

Random Reference Processing

Use of the random processing method generally assumes that the input data (transaction records) comes from a source other than disk storage, although a sequential disk file might be used for input. It also assumes that the transaction data is to be processed against data obtained from a disk file. The processing creates updated or new data to be placed back in disk storage or used as output to other input/output devices. Each transaction record contains the identification for one or more items in disk storage that are to be processed against the transaction data.

Separation of Routines

When a transaction record becomes available, the process of issuing Seeks and Reads for one or more disk records must be performed. To utilize this time it is convenient to divide the total processing of an item into smaller routines, which will be referred to as the main routine and the disk routine.

The main routine is used to obtain transaction records and place them in an area of storage available to the disk routine. If the user desires, the main routine may include other operations that do not require data to be obtained by the disk routines.

Control is passed by the main routine to the disk routine after a transaction record has been made available by the main routine, and the disk routine proceeds to obtain the disk record needed, process it against the transaction record, and prepare the result for placement back into disk storage and/or as output data for some other input/output device. Because of the time required for disk functions, i.e., Seek and Read, between the execution of a main routine and the ability of the disk routine to process a record, a void exists during which no instructions referring to the data may be executed. Random processing schemes utilize this time to read in more transaction records or, if possible, initiate execution of another disk routine for which data is available in core storage. Thus, in addition to keeping the computer busy, new requests for disk data are being generated continually so that access mechanisms are being utilized to the fullest extent with overlapped Seek operations.

If the user requires the output from a program to be in the same sequence as the transaction records that are brought in, an option is provided whereby the user may force the execution of disk routines in the same order as the transaction records are brought into the computer.

The Random-Processing Scheduler

Relationship to the Operating System

The Scheduler consists of routines that are incorporated into the using program by means of the Linkage Loader. The Scheduler routines are *not* contained in the Resident IOCS. The Scheduler routines are placed in relocatable format on a library file, which, when the routines are to be incorporated in a program, must be assigned as the *System Library file*. Information concerning the placement of Scheduler routines on the library file, and assignment of a System Library file, is contained in the publications, *IBM 1410/7010 Operating System; System Generation*, Form C28-0352, and *IBM 1410/7010 Operating System; System Monitor*, Form C28-0319, respectively. In brief, the assembly of a program utilizing the Scheduler routines takes the following steps:

1. The output of the Autocoder processor is one or more subprograms in relocatable format, containing imbedded calls for appropriate Scheduler routines. The calls are generated when the Autocoder processor encounters a Scheduler macro-instruction.

2. The subprograms are then processed into absolute format by the Linkage Loader, which gathers the calls for Scheduler routines, obtains the routines from the Library, and processes them into absolute format with linkage to the using program. The program, with Scheduler routines, is now in absolute format on the *Job file*.

The Scheduler imposes no requirements for control information beyond those stated in the *System Monitor* and *Autocoder* publications for assembly and execution of a program.

NOTE: Programs servicing a TELE-PROCESSING® System may be utilizing Scheduler routines and may be resident in core storage at the same time as a mainline program using the Scheduler. The result of this is the presence in storage of two sets of Scheduler routines. The Scheduler routines must not be incorporated in the Resident Monitor.

Relationship to the Basic Input/Output Control System

The Scheduler extends the Basic Input/Output Control System (IOCS) component of the Operating System by providing routines for handling disk data in a random processing application. These routines, combined with IOCS operations, will efficiently handle the input/output requirements of any random processing application. For example, in obtaining information from disk storage, the Scheduler determines whether or not a Seek operation is required, and in which input/output area the information is to be placed. Control is then passed to the IOCS to perform the actual input/output operations of seeking (if needed), reading the record, and handling error situations if they arise. In brief, the Scheduler routines combined with the IOCS will automatically:

Schedule the maximum number of Seek, Read, and Write operations for efficient operation
Overlap disk input/output operations with processing
Perform the actual Seek, Read, and Write operations
Check for read and write errors
Correct all correctable read and write errors

Capabilities of the Scheduler

Retention of Transaction Records

Successive transaction records are stored in a *Transaction Stacking Area* until they can be processed. If the Stacking Area is full, the Scheduler delays reading in any more transaction records until room is available. During this delay, disk records will be processed as they become available. The processing of a disk record takes precedence over reading another transaction record. Control is passed back to the main routine only if no disk record is available for processing and the Scheduler is able to accept more transaction records. (If two transaction records request the same disk record, the Scheduler does not process the second request until the first has been completed.) The Scheduler retains transaction records until notification that all disk routines needing the record have been satisfied.

Holding Disk Records

The records obtained by the read operations in the disk routines are held in the input/output area designated for that file until they are no longer required.

Correlation of Disk Records with Transaction Records

As described in the "Random Processing Concepts" section, the separation of routines permits concurrent handling of several input/output operations. Each time a disk record is successfully sought and read, it is made ready for processing. The Scheduler ensures that, whenever control is passed to a disk routine for processing a record, data from the *correct* transaction record is used.

Release of Transaction Stacking Areas and Input/Output Areas

The Scheduler ensures that no areas arc released to receive new data until all routines needing their present contents arc satisfied.

The Random-Processing Scheduler Macro-Instructions

The following is a brief description of each of the three macro-instructions used in coding a program utilizing the Scheduler facilities. Refer also to Figure 1, which shows graphically the placement of the instructions within a random processing scheme.

GET — Get Disk Record
This macro-instruction is used to SEEK (if necessary) and READ disk records.

PUT — Put Disk Record
This macro-instruction is used to SEEK (if necessary) and WRITE disk records.

IOCTL XXXXX — Input/Output Control
This macro-instruction can direct the Random-Processing Scheduler to perform a specific function as determined by the first operand (called the directing operand). Since the IOCTL operation cannot be used without the directing operand, the two will be referred to together as a form of the macro-instruction. The following are the forms of the macro-instruction, with a description of each.

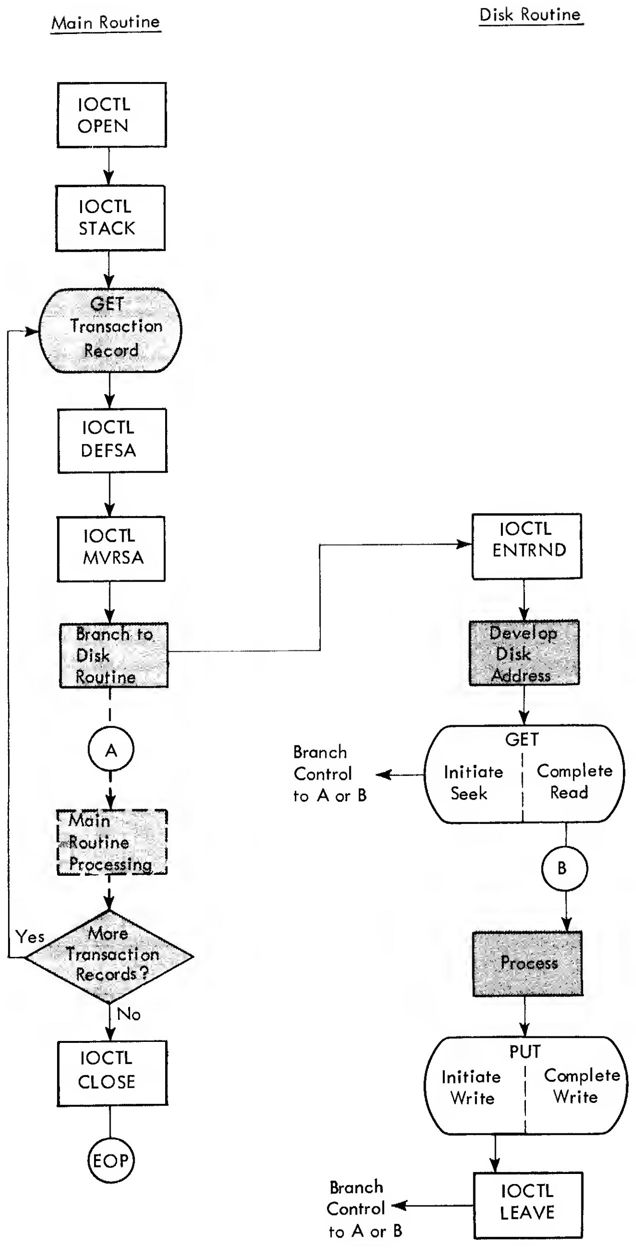


Figure 1. Macro-Instructions for Random Processing

IOCTL FSEQP — Force Sequential Processing.
This form of the macro-instruction is used to ensure that records are processed in the order in which transaction records were received, regardless of the order in which the disk records were accessed. It is not shown in Figure 1.

IOCTL MVRSA — Move Record to Stacking Area.
This form of the macro-instruction is used to move transaction records to the Transaction Stacking Area.

IOCTL OPEN — Open Disk Files.
This form of the macro-instruction is used to open disk files used for random processing.

IOCTL CLOSE — Close Disk Files.
This form of the macro-instruction is used to close disk files used for random processing.

IOCTL ENTRND — Enter Disk Routine.
This form of the macro-instruction must be used to define the beginning of any disk routine.

IOCTL LEAVE — Leave Disk Routine.
This form of the macro-instruction must be used to define the end of any disk routine.

IOCTL STACK — Open Stacking Area.
This form of the macro-instruction is used to establish linkage between the Transaction Stacking Area and the Scheduler. (Do not confuse this macro-instruction with the UNCTL STACK macro-instruction for basic IOCS.)

IOCTL DEFSA — Define Stacking Area.
This form of the macro-instruction is used to define a field in storage as a *temporary* Transaction Stacking Area.

Random files named in the operand field of a Scheduler macro-instruction must be defined by a DTF (Define the File) statement, written as described in the following section of this publication. Further, files *other than the files to be handled as random files* must be defined by DTF statements as specified in the publication, *Basic Input/Output Control System,* and the basic IOCS forms of the macro-instructions are used with them. For example, the transaction record file and its associated GET macro-instruction would be defined and used as detailed in the *Basic Input/Output Control System* publication. The OPEN and CLOSE functions for the transaction record file would utilize basic IOCS routines also.

As shown in Figure 1, the IOCTL OPEN macro-instruction is mandatory for each file, and directs the Scheduler to establish linkage with each file named in the operand. This macro-instruction also identifies a file as a RANDIN or RANDOUT file. (For definition of these terms see the PUT macro-instruction discussion in the next section of this publication.)

The IOCTL STACK macro-instruction establishes linkage between a Transaction Stacking Area and the Scheduler before the area is used by the program. This area is available to both the main routine and the disk routine and must be defined by a DA (Define Area) statement. (See the next section for details.)

The IOCTL MVRSA macro-instruction is used to place data (i.e., transaction records or data developed by the main routine) into a Transaction Stacking Area. It has three formats and will either: (1) move data from an input area to a segment of the Transaction Stacking Area; (2) select a segment of the Transaction Stacking Area, leaving the selection and movement of data to the user; or (3) will move data from an input area to one of several Transaction Stacking Areas.

The IOCTL ENTRND macro-instruction must be used as the first entry in a disk routine. It stores the return address to the main routine.

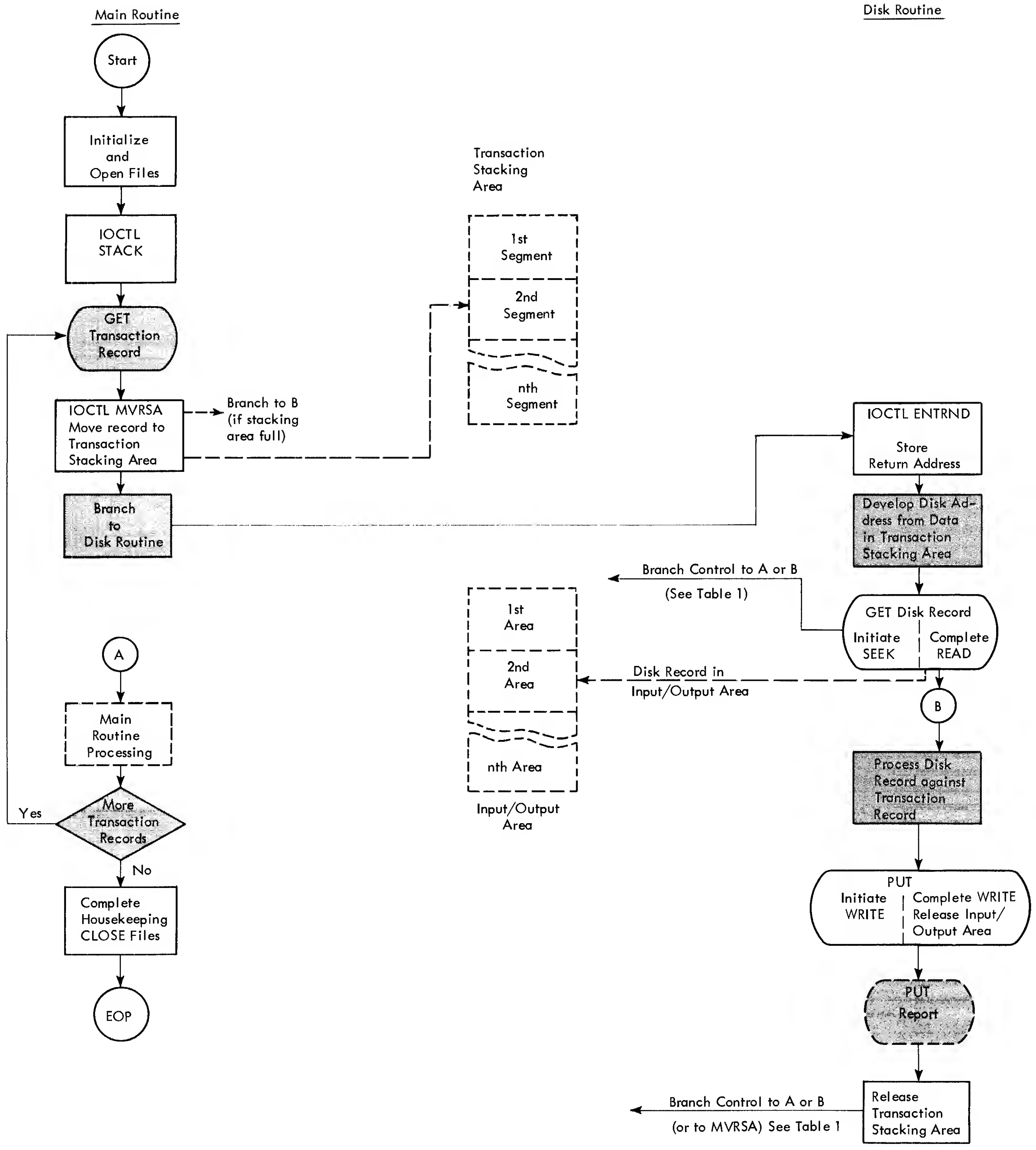
The GET macro-instruction makes a random file record available for processing. The user must develop the address needed to obtain the record.

The PUT macro-instruction is used to place processed records on a disk file, and will either place a record back in the location from which it was obtained, or place it in a location designated by the user.

The IOCTL LEAVE macro-instruction must be the last instruction used in a disk routine. It handles the release of the input/output and Transaction Stacking Areas used by the disk routine with which it is associated. The LEAVE macro-instruction also performs checking functions, which are detailed in the next section.

The IOCTL DEFSA macro-instruction is closely related to the MVRSA instruction. It allows the Scheduler to treat the location of any data in storage as a segment of a Transaction Stacking Area. The user must provide a routine to release this *temporary* transaction area.

The IOCTL FSEQP macro-instruction is not shown in Figure 1. Details of its operation are discussed in the next section.



Main Routine
Start
Initialize and Open Files
IOCTL STACK
GET Transaction Record
IOCTL MVRSA Move record to Transaction Stacking Area
Branch to B (if stacking area full)
Branch to Disk Routine
A
Main Routine Processing
More Transaction Records
Yes
No
Complete Housekeeping CLOSE Files
EOP
Transaction Stacking Area
1st Segment
2nd Segment
nth Segment
1st Area
2nd Area
nth Area
Input/Output Area
Disk Routine
IOCTL ENTRND Store Return Address
Develop Disk Address from Data in Transaction Stacking Area
Branch Control to A or B (See Table 1)
GET Disk Record
Initiate SEEK
Complete READ
Disk Record in Input/Output Area
B
Process Disk Record against Transaction Record
PUT
Initiate WRITE
Complete WRITE Release Input/Output Area
PUT Report
Release Transaction Stacking Area
Branch Control to A or B (or to MVRSA) See Table 1

Figure 2. Separation of the Main and Disk Routines

Separation of Processing Routines

The operating principles of the Scheduler are illustrated in Figure 2 and discussed in Table I. Note that all instructions needed to obtain or process disk data have been removed from the main routine. Transaction records are stored in a special area (Transaction Stacking Area) to await processing.

Separating the disk routine(s) from the main routine allows the Random-Processing Scheduler to use the waiting time to analyze the situation at that point. As a result, the Scheduler is able to coordinate the maximum possible processing in the disk routine(s).

Although the main routine initiates processing of the disk routine, the two routines are independent of one another: the main routine obtains and stores transaction records independently of any processing in the disk routine, and the disk routine obtains disk records and associated transaction records independently of processing in the main routine. The Scheduler coordinates the transaction record with the correct disk record.

Sequence of Disk Operations

Although transaction records are stacked in the order in which they were obtained by the main routine, updated disk records are not necessarily read from nor written back onto the disk in the same order. This is because of the different access times for information in disk storage. Information access time depends not only on the order in which disk requests are given, but also on the location of the requested information in disk storage. The Scheduler will schedule all requests so that the record that *can be* obtained first *is* obtained first.

For example, if the Scheduler receives a request which requires an access time of 100 ms, and if before this request is honored by the Scheduler, another request is received that requires the same access mechanism but only requires an access time of 50 ms, the *last* request will be scheduled ahead of the *first* request. In another example, if one access mechanism receives a request for information which requires an access time of 100 ms, and 10 ms later another access

Main Routine	Disk Routine
The main routine obtains a transaction record and requests stacking of the record in a segment of the Transaction Stacking Area. If a segment is available, the record is stacked and control is given to the disk routine. If no segment is available for the transaction record, processing will be permitted only in the disk routine until a segment is released. At this time, processing in the main routine will resume.	
	Processing in the disk routine proceeds until a request for a GET operation is encountered. The disk routine initiates the SEEK and then checks all input/output areas. If a previously read disk record is ready for updating, processing continues in the disk routine (Point B). If no disk record is ready for processing in an input/output area and the Scheduler is able to accept additional random requests, control is given to the main routine at Point A.
Processing now continues at Point A (return address of the main routine). Main routine processing (if any) is carried out, and a branch to the instruction that calls for the reading of the next transaction record is taken. This transaction record is moved to a free segment of the Transaction Stacking Area and control is given to the disk routine at Point B.	The waiting disk record is updated with the correct transaction data, and the WRITE operation that will write the updated disk record back into disk storage is initiated. (As indicated in Figure 2, the input/output area that contained the just-updated disk record will be released upon completion of the WRITE operation.) When processing in the disk routine continues, any needed report is written, and control of the segment of the Transaction Stacking Area that held the correct transaction record is returned to the Random-Processing Scheduler. The check for an input/output area ready for updating is then made again, and control is given to Point A or Point B, depending on the outcome of the check.

Table I. Program Steps Executed by the Random-Processing Scheduler

mechanism receives a request which requires access time of 50 ms, again the second request will be met before the first. In either case, the disk information requested last will be obtained before that information requested just prior to it.

If disk records must be processed in the same order as the incoming transaction records, processing of the data obtained by the second request in the examples must be delayed until the data obtained by the first request has been processed. This is accomplished by the use of the IOCTL FSEQP (Force Sequential Processing) macro-instruction. (See the next section for details of the FSEQP macro-instruction.)

Independent Disk Routines

Disk routines are considered independent when the processing in one routine is not supported by the processing in another disk routine.

For example, some applications require two (or more) *independent* disk routines. A typical application of this type is the updating of a job record *and* an employee record on the basis of one transaction record. Both routines use the same Transaction Stacking Area. However, each routine operates on the transaction record independently, not requiring data developed by the other routine (Figure 3).

The Scheduler ensures that no segment of the Transaction Stacking Area is released until all disk routines requiring data from this segment have been completed. The Scheduler can handle any number of independent disk routines.

Dependent Disk Routines

Some applications use data from one transaction record to update two *dependent* disk records. Two records are considered dependent on one another if neither of them can be updated without data from the other. When dependent records are processed, *both* disk records must be obtained before either of them can be updated. This requires that the disk records be retained in their input/output areas until *all* disk records accessing the related data have been completed. The Scheduler can handle any number of dependent disk routines.

Dependent disk routines require the use of the IOCTL FSEQP macro-instruction. In Figure 4, the IOCTL FSEQP macro-instruction in Disk Routine A retains the disk record obtained, so that the record is available for use by Disk Routine B. The IOCTL FSEQP macro-instruction in Disk Routine B holds up further processing in the routine until the disk records requested by both routines (A and B) are made available. (For a detailed description of the functions of the IOCTL FSEQP macro-instruction, see the section "Additional Functions of the IOCTL FSEQP Macro-Instruction.")

Programming Considerations

The following material is offered as an aid to the programmer planning a program utilizing the Scheduler facilities.

1. All random files to be referenced must be defined by a DTF statement as specified in this manual.

2. Transaction Stacking Areas and input/output areas must be defined by DA statements as specified in this manual.

3. The IOCTL MVRSA, IOCTL STACK, IOCTL OPEN, IOCTL CLOSE, and IOCTL DEFSA macro-instructions are used in the main routine.

4. A GET macro-instruction appearing in the main routine cannot reference a file that has been defined as a *random* file.

5. No branching between disk routines may occur.

6. A disk routine must begin with an IOCTL ENTRND macro-instruction and end with an IOCTL LEAVE macro-instruction.

7. Within a disk routine, except for the macro-instructions covered in item 3, all IOCS macro-instructions may be used.

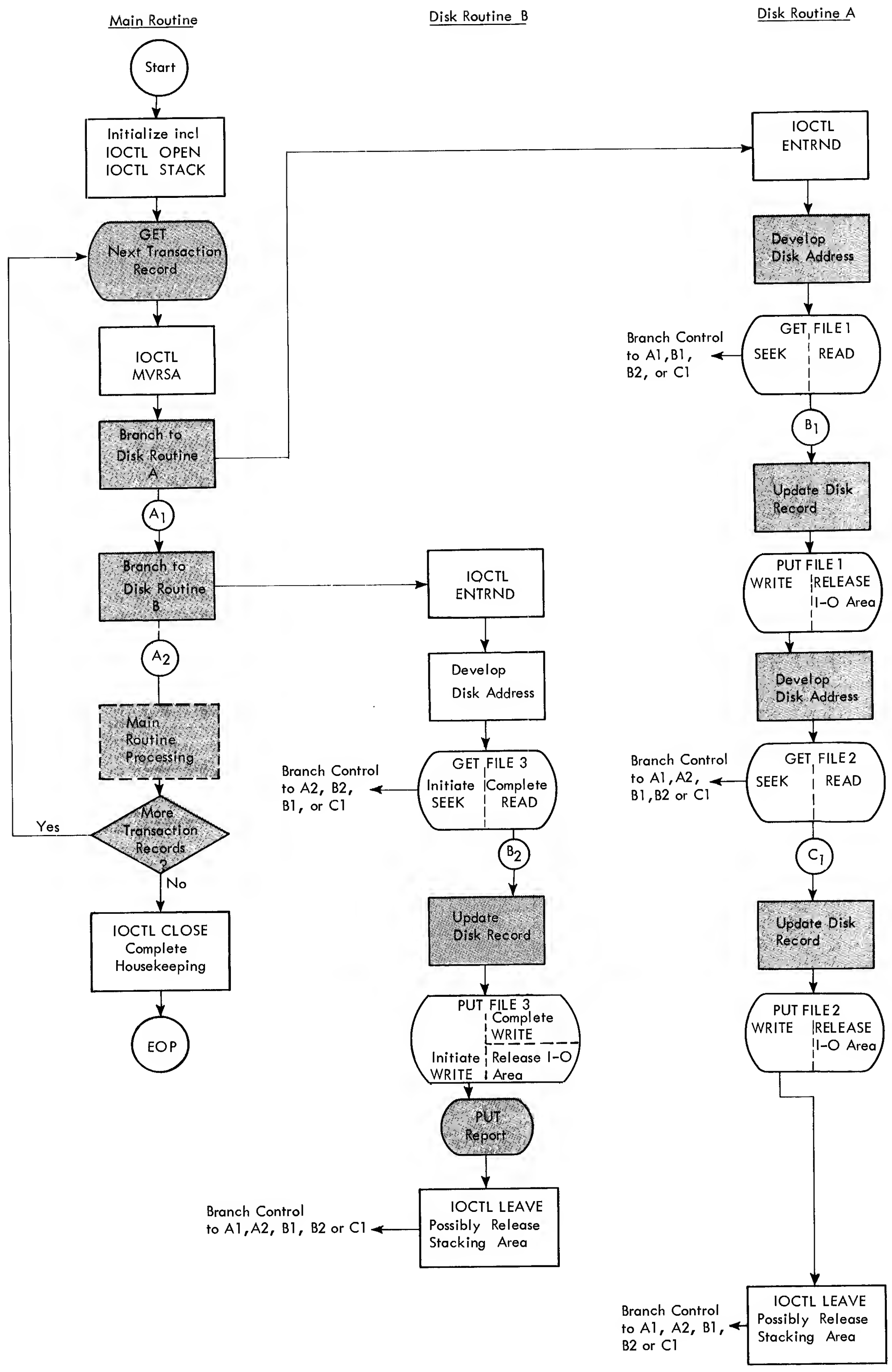
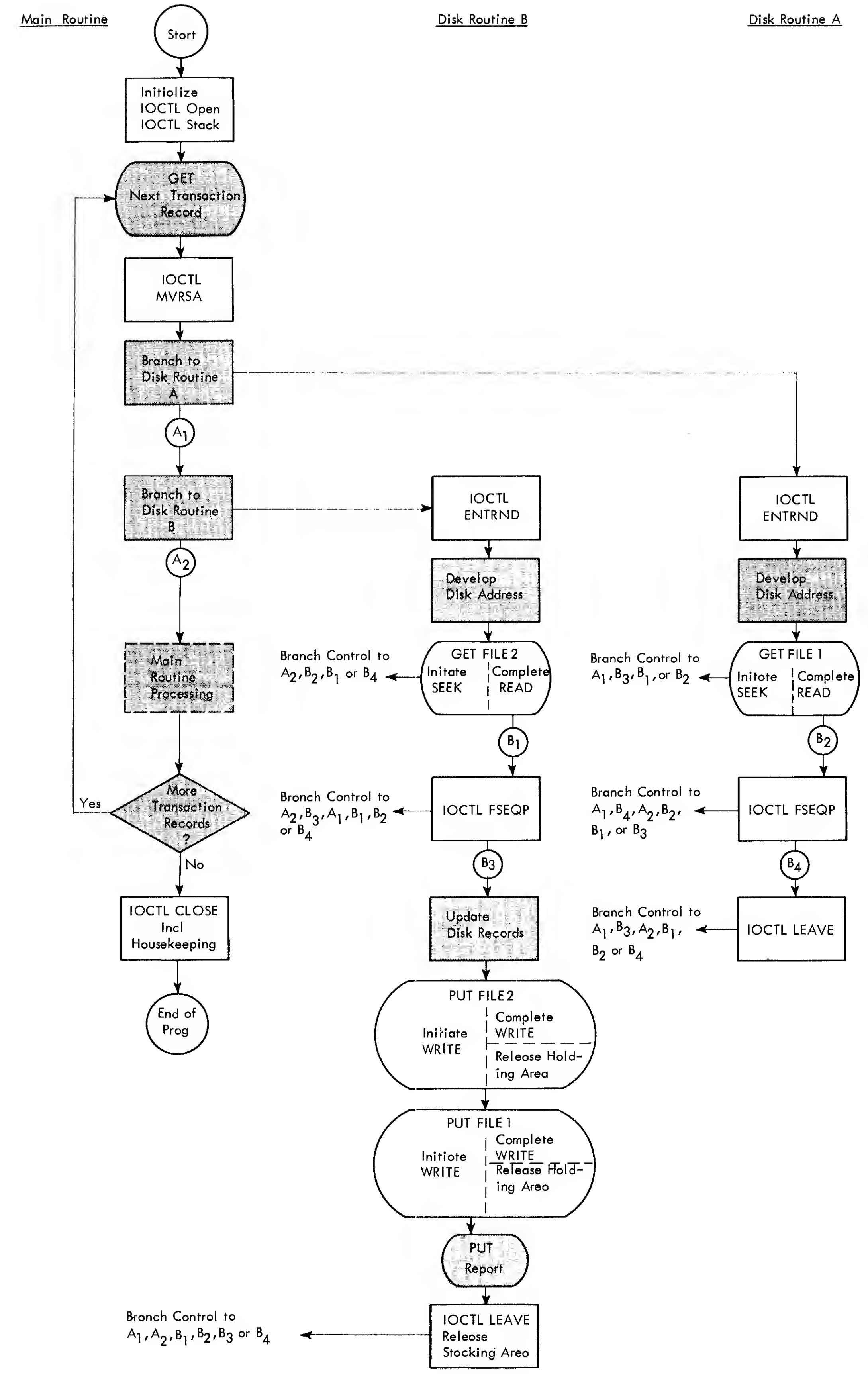


Figure 3. Random Processing – Two Independent Disk Routines Use Data Obtained by the Main Routine



Main Routine
Disk Routine B
Disk Routine A
Start
Initialize IOCTL Open IOCTL Stack
GET Next Transaction Record
IOCTL MVRSA
Branch to Disk Routine A
A1
Branch to Disk Routine B
A2
Main Routine Processing
More Transaction Records ?
Yes
No
IOCTL CLOSE Incl Housekeeping
End of Prog
IOCTL ENTRND
Develop Disk Address
GET FILE 2 Initiate SEEK | Complete READ
Branch Control to A2,B2,B1 or B4
B1
IOCTL FSEQP
Branch Control to A2,B3,A1,B1,B2 or B4
B3
Update Disk Records
PUT FILE 2 Initiate WRITE | Complete WRITE | Release Holding Area
PUT FILE 1 Initiate WRITE | Complete WRITE | Release Holding Area
PUT Report
IOCTL LEAVE Release Stacking Area
Branch Control to A1,A2,B1,B2,B3 or B4
IOCTL ENTRND
Develop Disk Address
GET FILE 1 Initiate SEEK | Complete READ
Branch Control to A1,B3,B1,or B2
B2
IOCTL FSEQP
Branch Control to A1,B4,A2,B2, B1, or B3
B4
IOCTL LEAVE
Branch Control to A1,B3,A2,B1, B2 or B4

Figure 4. Random Processing — Two Dependent Disk Routines Use Data Obtained from the Main Routine

This section describes in detail the function and use of the macro-instructions and DTF and DA statements that must be placed in a program making use of the facilities of the Random-Processing Scheduler.

Macro-Instructions

GET

The programmer uses the GET macro-instruction to make a disk record available for processing.

Before using this macro-instruction for random processing, the programmer must store the appropriate address (single-record or full-track) in an eight-character field within the Scheduler. The low-order position of this field is represented by the system symbol, /DKA/, as shown in Figure 5.

Channel	Number of Disk Unit	Track or Record Address					
@ or *							

/DKA/

Figure 5. Location /DKA/

In either single-record or full-track random processing, the user must store in /DKA/ the channel designation and the number of the disk module as indicated. The remainder of /DKA/ is either the six-character record address of the desired record or the track address.

If the single-record mode is used, the user must place the four-digit track address (HA1) into another field identified by the system symbol /DKS/. (This field is located within the Scheduler and is a four-character field immediately to the left of /DKA/.) The user must then place the record address in /DKA/.

In full-track operations /DKA/ must contain the four-character HA1 followed by the two-character HA2. Since, in full-track operations, there is no record address, the field /DKS/ is not used.

Each time the programmer uses the GET macro-instruction, the Scheduler, combined with the Basic IOCS, will develop the coding required to do the following.

1. Check whether another disk operation is using the disk track specified by the disk address in the /DKA/ or /DKS/ location.
2. Assign an available input/output area into which the disk record is to be read.
3. Assign an access mechanism to read the information.
4. Delay processing on this record if the required disk track is being used by another disk operation.
5. Seek the track specified by the disk address.
6. Read the disk record into the assigned input/output area.
7. Check for disk read errors.
8. Correct the correctable read errors.
9. Release the input/output area used by a GET macro-instruction preceding the present GET macro-instruction. (See the section describing the IOCTL FSEQP macro-instruction.)
10. Associate disk records with the appropriate segment of the Transaction Stacking Area.
11. Check whether another disk record is ready for processing in an input/output area.
12. Give control to the appropriate disk routine if another disk record is waiting to be processed; or give control to the main routine if the Scheduler is able to accept additional requests.

PUT

The programmer may use the PUT macro-instruction to develop the coding required to include processed or unprocessed records in a disk file.

For the purpose of this discussion, the PUT macro-instructions will be divided into two types: (1) that used for input files (to return updated records to disk storage), and (2) that used for output files (to load information into *nonsequential* locations in disk storage).

Returning Updated Records to Disk Storage: This type of PUT macro-instruction is used to return updated disk records to the locations in disk storage from which they were originally obtained. To use this type of PUT macro-instruction the file must be opened as a random *input* file. The macro-instruction is written as indicated in Figure 6.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1		PUT	ACCOUNTS
0 2			

Figure 6. The PUT Macro-Instruction, Type 1

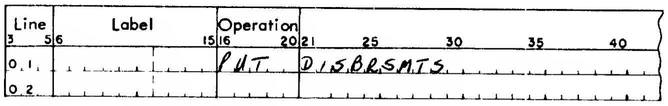
The operand in Figure 6 is the name of the disk file into which records are to be returned. The name must be that used to describe the file in the DTF Header Line.

When using this type of PUT macro-instruction, the programmer must process the disk records in input/output areas.

Loading Records into Nonsequential Disk-Storage Locations: This type of PUT macro-instruction is used to load records into nonsequential locations in disk storage. To use this type of PUT macro-instruction, the file must be opened as a random *output* file. (An example of this kind of application is the loading of new-parts records into an existing inventory file.) Before using this type of PUT macro-instruction, the programmer must:

1. Place the necessary addressing information in the locations /DKA/ and /DKS/. (See Figure 5 and the discussion of addressing in the description of the GET macro-instruction.)
2. Place the information to be written into disk storage into the output area that is addressed by the index register specified by the DTF INDEX entry of the file.

This type of PUT macro-instruction is written as illustrated in Figure 7.



Line 3–5	Label 6–15	Operation 16–20	21 25 30 35 40
0 1		PUT	DISBRSMTS
0 2			

Figure 7. The PUT Macro-Instruction, Type 2

The operand in Figure 7 is the name of the disk file into which records are to be loaded. The name is that used to describe the file in the DTF Header Line.

NOTE 1: If an output file is the only random file opened, this form of the PUT macro should *not* be contained in a disk routine.

NOTE 2: This type of PUT macro-instruction (with an output file named in the operand) cannot be used to return an updated disk record to the location in which it was originally contained.

NOTE 3: This type of PUT macro-instruction may cause the replacement of the entire contents of a disk track, according to the record format used. The programmer is cautioned against inadvertently destroying disk data when using this macro-instruction. (The data will be written on disk as specified by the format track; therefore, when writing records that are shorter than defined by the format track, the remainder of the record will be padded with blanks.)

IOCTL OPEN

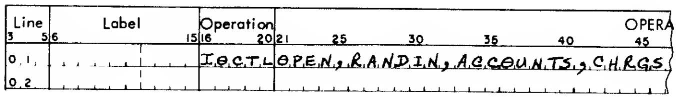
This macro-instruction is mandatory for each file. It directs the Scheduler to establish linkage with the file. The operands are:

OPEN — This operand is the directing operand.

RANDIN (or RANDOUT) — This operand describes a random input (or random output) file. Only one of these operands may be present in each IOCTL OPEN macro-instruction.

FILENAME — This operand contains the name of the file to be opened. The name must be the same one used in the DTF Header Line. There may be one or more FILENAME operands per IOCTL OPEN macro-instruction; however, input and output files cannot be mixed.

Figure 8 shows the files named ACCOUNTS and CHRGS being opened as random input files.



Line 3–5	Label 6–15	Operation 16–20	OPERA 21 25 30 35 40 45
0 1		IOCTL	OPEN,RANDIN,ACCOUNTS,CHRGS
0 2			

Figure 8. The IOCTL OPEN Macro-Instruction

IOCTL CLOSE

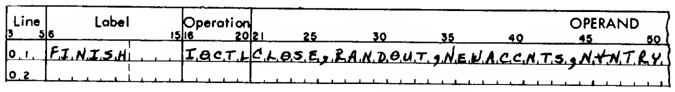
This macro-instruction is mandatory for each file. It directs the Scheduler to remove the named random file from use by the Scheduler. The closing function will not take place until the Scheduler has honored all requests that were pending at the time the CLOSE form of the macro-instruction was encountered. The operands are:

CLOSE — This is the directing operand.

RANDIN (or RANDOUT) — This operand describes a random input (or random output) file. Only one of these operands may be present.

FILENAME — This operand contains the name of the file to be closed. The name must be the same one used in the DTF Header Line. There may be one or more FILENAME operands per IOCTL CLOSE macro-instruetion; however, input and output files cannot be mixed.

Figure 9 shows the files named NEWACCNTS and NVNTRY being closed as random output files.



Line 3–5	Label 6–15	Operation 16–20	OPERAND 21 25 30 35 40 45 50
0 1	FINISH	IOCTL	CLOSE,RANDOUT,NEWACCNTS,NVNTRY
0 2			

Figure 9. The IOCTL CLOSE Macro-Instruction

IOCTL STACK

This form of the IOCTL macro-instruction is used to establish linkage between a Transaction Stacking Area and the Scheduler *before the area is used by the program.* Each Transaction Stacking Area contained within the program must be linked to the Scheduler by this macro-instruction. The four operands described below are required, and must be written in the order shown in Figure 10. They are:

STACK — This is the directing operand.

LABEL — The second operand is the label of a Transaction Stacking Area defined by the user in a DA statement. (See the section "DA Entries Needed to Support the Random-Processing Scheduler.")

LENGTH — The third operand states the length of a segment of this Transaction Stacking Area. All segments of any stacking area must be the same length and must include a position for a group mark with word mark or a record mark. (See the section "DA Entries Needed to Support the Random-Processing Scheduler.")

INDEX — This operand specifies the index register (X1, X2 . . . . . X12) to be assigned to this Transaction Stacking Area. When the Scheduler returns to the user at GET+1, the specified index register will contain the address of the current Transaction Stacking Area. If the IOCTL STACK macro-instruction appears several times in a program, the same index register must be specified each time.

NOTE: Index registers 13, 14 and 15 must not be used for this purpose.

The operand in Figure 10 informs the Scheduler that the Transaction Stacking Area labeled TRAREA contains segments of 81 positions (80 data positions and 1 position for a group mark with word mark or a record mark), and that the user refers to the area through index register 8.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1		IOCTL	STACK,TRAREA,81,X8
0 2			

Figure 10. The IOCTL STACK Macro-Instruction

IOCTL MVRSA (Move Record to Stacking Area)

All data developed by the main routine and required by the disk routine(s) must be placed into a Transaction Stacking Area. This ensures that the main routine does not alter or destroy the data before it has been used by the disk routine(s).

The programmer may use the IOCTL MVRSA macro-instruction to transfer data developed in the main routine to a segment of the Transaction Stacking Area specified by an IOCTL STACK macro-instruction.

The data will be retained in the Transaction Stacking Area until all disk operations using the data have been completed. If the Stacking Area is full, the Scheduler will wait until processing in a disk routine makes a Stacking Area segment available. The IOCTL MVRSA form of the macro-instruction must be given in the main routine before control is branched to the disk routine. See Figure 11. There are three formats of this macro-instruction.

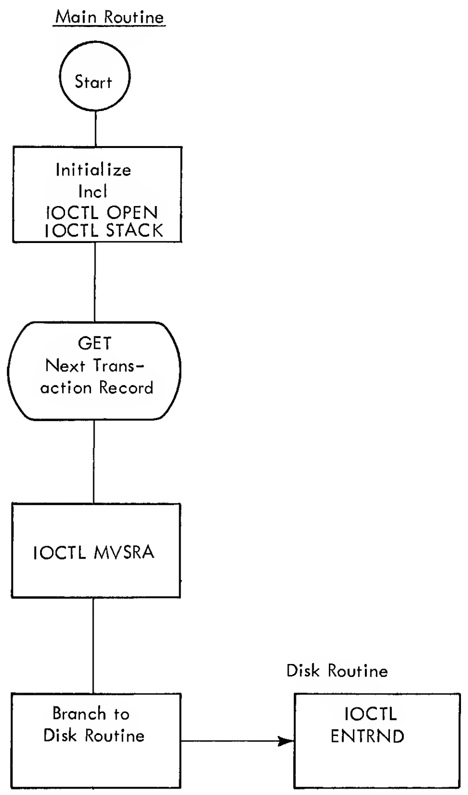


Figure 11. Use of the IOCTL MVRSA Macro-Instruction

FORMAT A

Format A has two operands and is written as indicated in Figure 12. The second operand, which may be indexed, identifies the high-order position of the area from which information is to be moved to the Transaction Stacking Area. Each area from which information is to be removed must have a record mark or a

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1	ANYLABEL	IOCTL	MVRSA,INFOLABEL
0 2			

Figure 12. The IOCTL MVRSA Macro-Instruction, Format A

group mark with word mark immediately to the right of the low-order position. See Figure 13.

This format of the IOCTL MVRSA macro-instruction will cause the Scheduler to:

1. select an available segment of the Transaction Stacking Area specified by an IOCTL STACK macro-instruction,
2. insert the address of the selected segment into the index register specified by the IOCTL STACK macro-instruction, and
3. move the information and the word marks contained in the area specified by the operand into the segment of the Transaction Stacking Area selected by the Scheduler.

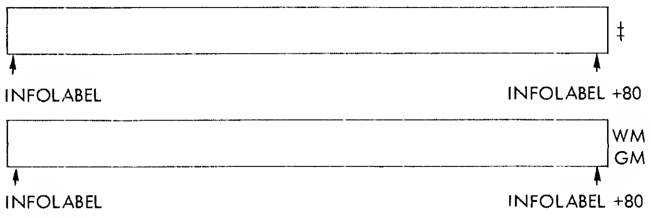


Figure 13. The 80-Character Area Referred to by the Second Operand of the IOCTL MVRSA Macro-Instruction

FORMAT B

Format B of the IOCTL MVRSA macro-instruction has one operand and is written as indicated in Figure 14.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1	ANYLABEL	IOCTL	MVRSA
0 2			

Figure 14. The IOCTL MVRSA Macro-Instruction, Format B

This format of the IOCTL MVRSA macro-instruction *does not move* the transaction record, but only selects a Transaction Stacking Area segment. The user is responsible for moving the transaction record; therefore, the user is afforded the flexibility of moving only the portion of the transaction record that contains significant information. The user may code the B-address of the Move instruction relative to zero and index with the index register specified by the IOCTL STACK macro-instruction.

This format of the IOCTL MVRSA macro-instruction will direct the Scheduler to:

1. select an available segment of the Transaction Stacking Area specified by the IOCTL STACK macro-instruction, and
2. insert the address of the selected section of the Transaction Stacking Area into the index register specified by the IOCTL STACK macro-instruction.

FORMAT C

Format C of the IOCTL MVRSA macro-instruction has three operands. It is written as indicated in Figure 15.

Line 3 5	Label 6 15	Operation 16 20	OPER 21 25 30 35 40 45
0 1	LABEL	IOCTL	MVRSA,INFOLABEL,TRAREA
0 2			

Figure 15. The IOCTL MVRSA Macro-Instruction, Format C

This form of the IOCTL MVRSA macro-instruction directs the storing of data into one of several Transaction Stacking Areas. In this way a program may contain more than one Transaction Stacking Area that must be defined by an IOCTL STACK macro-instruction. The use of these Transaction Stacking Areas is dependent upon the operands of this format of the IOCTL MVRSA macro-instruction.

The first operand is the directing operand.

The second operand identifies the high-order position of the area from which information is to be moved to the Transaction Stacking Area. This operand may be indexed.

The third operand identifies the Transaction Stacking Area into which the transaction record is to be stored. This operand is the identifying label of the Transaction Stacking Area. (See the section "DA Entries Needed to Support the Random-Processing Scheduler.")

This format of the IOCTL MVRSA macro-instruction will cause the Scheduler to:

1. select an available segment of the Transaction Stacking Area designated by the third operand (TRAREA) of this macro-instruction and defined by an IOCTL STACK macro-instruction,
2. insert the address of the selected segment into the index register specified by the IOCTL STACK macro-instruction, and
3. move the information and the word marks from the area specified by the second operand (INFOLABEL) to the segment of the Transaction Stacking Area specified by the Scheduler.

All Transaction Stacking Areas must be initiated with an IOCTL STACK macro-instruction, and all IOCTL STACK macro-instructions must refer to the same index register.

IOCTL DEFSA — Define Stacking Area

This form of the IOCTL macro-instruction is very closely related to the IOCTL MVRSA macro-instruction (Figure 16). This macro-instruction will allow the Scheduler to treat the address of any area in core storage as a segment of a Transaction Stacking Area. Whereas the

IOCTL MVRSA macro-instruction is used to pick an available segment from a predefined Transaction Stacking Area, the IOCTL DEFSA macro-instruction is used to handle the area addressed by the index register specified in the second operand as a temporary Transaction Stacking Area segment. Even though the function of the IOCTL MVRSA macro-instruction is closely approximated, the data in the temporary area is not moved, and may not be altered or destroyed until control is given to the routine labeled by the third operand. The operands are:

DEFSA — This is the directing operand.

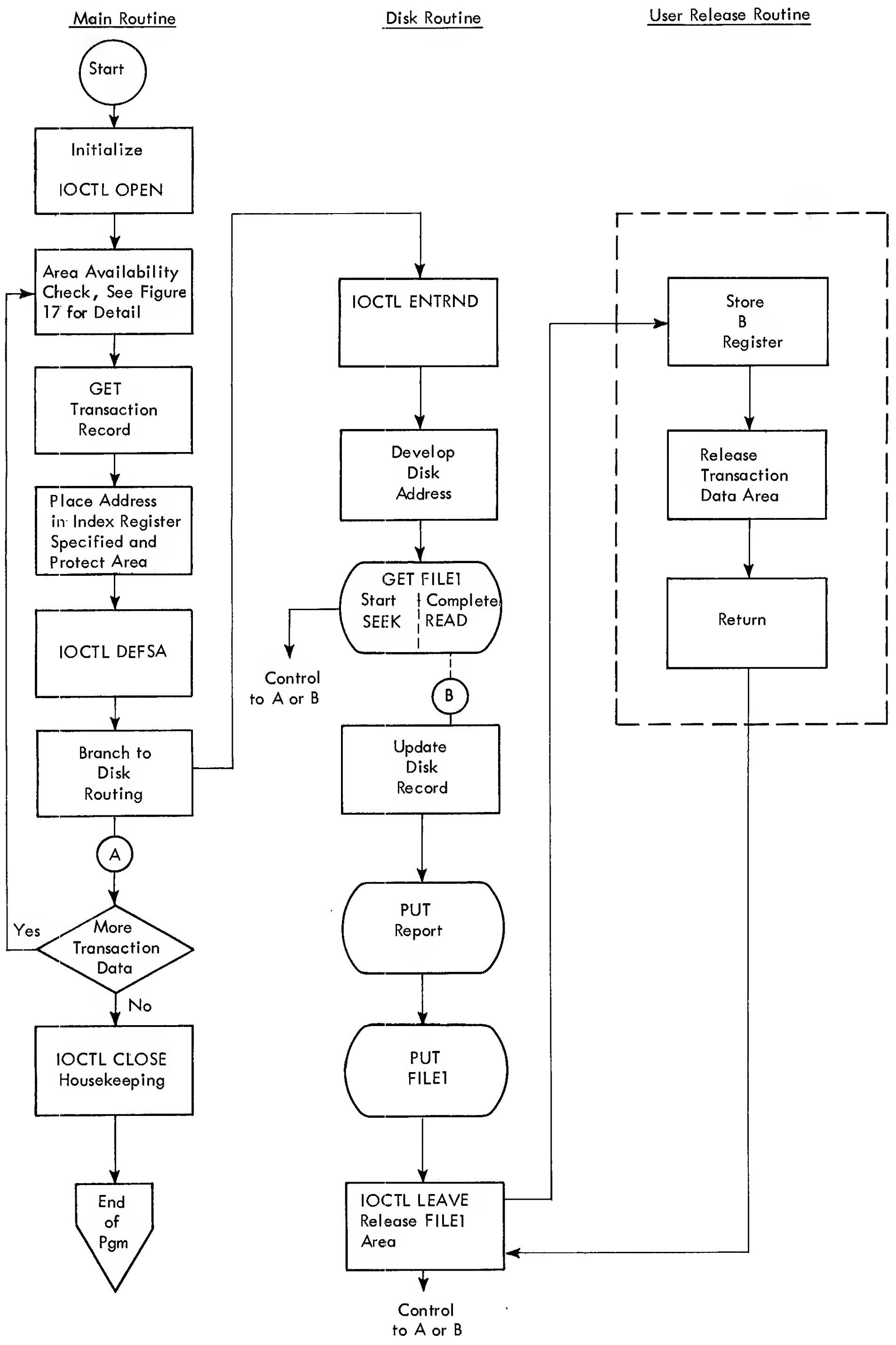


Figure 16. The IOCTL DEFSA Macro-Instruction Used in a Random Processing Scheme

INDEX – The index register specified should contain an address (of an area) which will be associated with a disk record as transaction data. The Scheduler saves the contents of the index register and restores the register when the requested disk record is available. The user supplies this address.

LABEL – This operand must contain the label of the user-written closed subroutine to which the Scheduler will give control when the temporary Stacking Area is ready to be released. At the time this exit is taken, the Scheduler will place into the index register specified in the second operand the address of the area which is no longer required. The closed subroutine must not contain any Scheduler macro-instructions or any macro-instructions that may require the Scheduler.

The user-written routine must first store the re-entry point to the Scheduler (thus, a *closed* subroutine), perform whatever function is necessary to properly release the temporary area, and return to the entry point of the Scheduler.

SPECIAL CONSIDERATIONS

The Scheduler does not protect areas which are stacked by this macro-instruction. If this area is an input/output area, the associated IORW (Input/Output Request Word) should be removed from the file before the IOCTL DEFSA macro-instruction is given. The IORW may be returned to the file by the user's *closed* subroutine.

Figure 17 illustrates a suggested method for handling a possible situation in which no areas are free to be reassigned by the DEFSA macro-instruction: Test for area availability. If no area is free, branch to a dummy disk routine which contains only the IOCTL ENTRND and IOCTL LEAVE macro-instructions. Upon returning to the main routine, repeat the test for area availability.

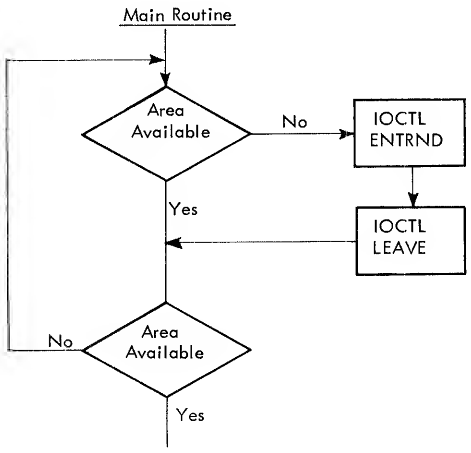


Figure 17. Suggested Wait Routine

If indexing is used to refer to fields within a temporary Stacking Area, the index register used must be the same one specified in the IOCTL DEFSA macro-instruction.

The IOCTL DEFSA macro-instruction will:

1. direct that the area addressed by the index register be treated as a Transaction Stacking Area associated with all subsequent random requests until another IOCTL DEFSA or IOCTL MVRSA macro-instruction is encountered,

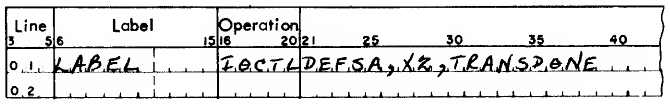
2. retain the address of the closed subroutine to be entered when the transaction area is no longer required, and

3. restore to the specified index register the data placed there by the user (prior to issuing the IOCTL DEFSA macro-instruction), and enter the release subroutine when the transaction area is ready to be released.

The macro-instruction in Figure 18 will cause the Scheduler to treat the area addressed by the contents of index register 2 as a segment of the Transaction Stacking Area. Control will be given to the routine labeled TRANSDONE, when the area is no longer required.

NOTE 1: This macro-instruction must always be given in the main line of the program.

NOTE 2: The functions performed by the Scheduler for the use and release of temporary Stacking Areas are based on the assumption that at least two different areas will be used by DEFSA macro-instructions.



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
01	LABEL	IOCTL	DEFSA,X2,TRANSDONE
02			

Figure 18. The IOCTL DEFSA Macro-Instruction

IOCTL ENTRND

The IOCTL ENTRND macro-instruction must be the first instruction used in any disk routine of a program using the Scheduler. This macro-instruction develops the coding required to store the return address of the main routine. This is the address to which control will be returned by the Scheduler to continue processing the main routine. See Figure 19. The user may refer to the label of this macro-instruction (Figure 20) in branching to his disk routine.

The coding provided by the IOCTL ENTRND macro-instruction stores the return address of the main routine. This permits resumption of processing in the main routine as soon as the next disk operation has been initiated.

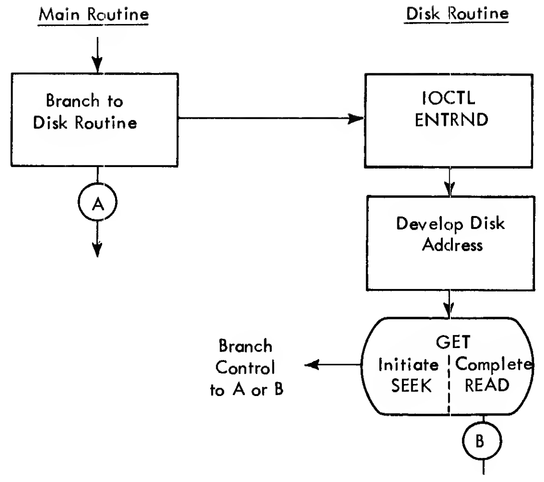


Figure 19. Use of the IOCTL ENTRND Macro-Instruction

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1	ANYLABEL	IOCTL	ENTRND
0 2			

Figure 20. The IOCTL ENTRND Macro-Instruction

IOCTL FSEQP

The programmer may use the IOCTL FSEQP macro-instruction to ensure that disk records obtained by the disk routine(s) will be processed and returned to disk storage in the same order in which the corresponding transaction records were obtained by the main routine (Figure 21).

For example: Assume that Module 1 receives a request for information with an access time of 100 ms, and that 10 ms later Module 2 receives a request with an access time of 50 ms. In this case, the access mechanism of Module 2 will obtain the specified information before the access mechanism of Module 1.

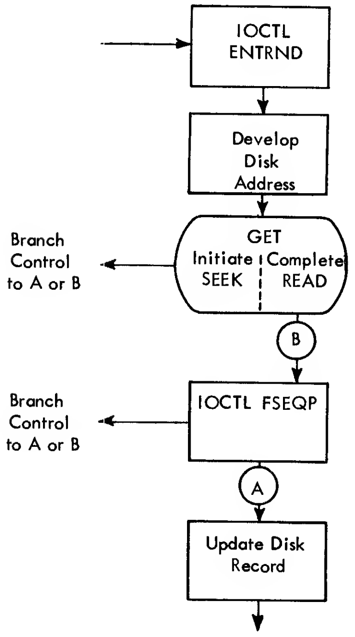


Figure 21. Use of the IOCTL FSEQP Macro-Instruction

If the disk routine does not use the IOCTL FSEQP macro-instruction, the information obtained by the access mechanism of Module 2 will be processed and the result returned to disk storage before the information sought by the access mechanism of Module 1 can be read. In this case, the updated information will not be returned to disk storage in the order in which the corresponding transaction records were read by the main routine.

If the disk routine in the example uses the IOCTL FSEQP macro-instruction, the information obtained by the access mechanism of Module 2 will not be processed until the information obtained by the access mechanism of Module 1 has been processed.

The IOCTL FSEQP macro-instruction has one operand, and is written as indicated in Figure 22. It may be used anywhere in the program.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1	ANYLABEL	IOCTL	FSEQP
0 2			

Figure 22. The IOCTL FSEQP Macro-Instruction

Each time the programmer inserts an IOCTL FSEQP macro-instruction in his program, the Scheduler will develop the coding required to:

1. delay the processing of the disk record until all previously requested records have been processed,
2. branch control to the appropriate disk routine if another disk record is waiting to be processed, or
3. branch control to the main routine if the Scheduler is able to accept additional requests, or
4. branch control to a waiting loop if a disk record cannot be processed and if the Scheduler cannot accept additional random requests (Figure 23).

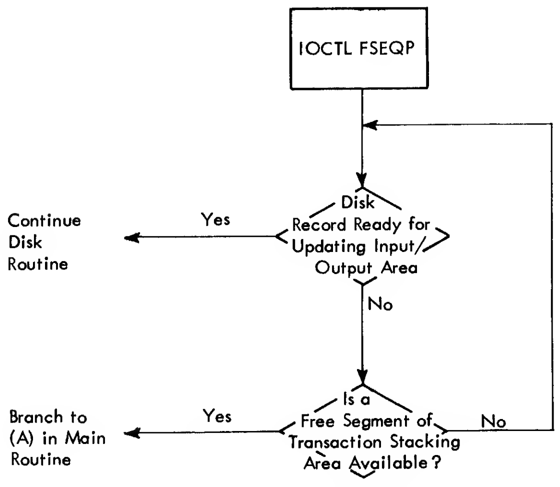

IOCTL FSEQP
Disk Record Ready for Updating Input/Output Area
Yes
Continue Disk Routine
No
Is a Free Segment of Transaction Stacking Area Available?
Yes
Branch to (A) in Main Routine
No

Figure 23. Control Functions of the IOCTL FSEQP Macro-Instruction

If the IOCTL FSEQP macro-instruction is given in the main routine, processing in the main routine *will not continue* until all random disk storage data has been processed.

ADDITIONAL FUNCTIONS OF THE IOCTL FSEQP MACRO-INSTRUCTION

The IOCTL FSEQP macro-instruction has two important additional functions. It can be used to *prevent:*

1. the release of an input/output area by the second of two successive GET macro-instructions, for different files, or
2. the release of an input/output area by an IOCTL LEAVE macro-instruction.

RETENTION OF DISK DATA AFTER SECOND GET MACRO-INSTRUCTION

Each time two disk GET macro-instructions follow one another in a program, the second GET macro-instruction causes the release of the information obtained by the first GET macro-instruction. Thus, the coding sequence

```
GET DISKFILE1
GET DISKFILE2
PROCESS
PUT DISKFILE2
```

will cause the release of the information obtained by the first GET macro-instruction, and only the information obtained by the second GET macro-instruction can be returned to disk storage by a PUT macro-instruction. The IOCTL FSEQP macro-instruction may be used to prevent the first GET macro-instruction, as indicated by the following coding sequence:

```
GET DISKFILE1
IOCTL FSEQP
GET DISKFILE2
PROCESS
PUT DISKFILE2
PUT DISKFILE1
```

In this case, the IOCTL FSEQP macro-instruction prevents the release of the information obtained by the first GET macro-instruction.

The order of the PUT macro-instructions should be given as indicated, since the access mechanism is already positioned at the track that contained the data from DISKFILE2.

The IOCTL FSEQP macro-instruction cannot be used to hold information obtained by the first of two GET macro-instructions involving the same file. Thus, in the coding sequence

```
GET DISKFILE1
IOCTL FSEQP
GET DISKFILE1
PROCESS
```

the IOCTL FSEQP macro-instruction cannot prevent the release of the information obtained by the first GET macro-instruction.

RETENTION OF DISK DATA AFTER THE IOCTL LEAVE MACRO-INSTRUCTION

The IOCTL LEAVE macro-instruction, described next, releases the information obtained by the GET macro-instruction(s) in the disk routine.

Thus the coding sequence

```
GET DISKFILE1
PROCESS
IOCTL LEAVE
```

will cause the release of the information obtained by the GET macro-instruction. The IOCTL FSEQP macro-instruction may be used to prevent the release of disk-record information by the IOCTL LEAVE macro-instruction. Thus, in the coding sequence

```
GET DISKFILE1
IOCTL FSEQP
IOCTL LEAVE
```

the information obtained by the GET macro-instruction will be retained in an input/output area for processing by a subsequent disk routine (Figure 4).

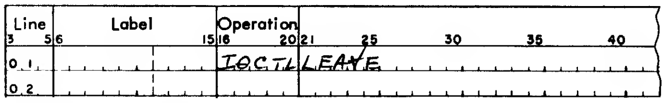
NOTE 1: An IOCTL FSEQP macro-instruction may require the Scheduler to reread a disk record; therefore, the contents of an input/output area must not be changed between the GET macro-instruction and the IOCTL FSEQP macro-instruction. The coding sequence:

```
GET DISKFILE1
UPDATE
IOCTL FSEQP
```

might result in rereading a disk record and overlaying the previous updating of the same record.

IOCTL LEAVE — Leave Disk Routine

The IOCTL LEAVE macro-instruction (Figure 24) must be the last instruction in a disk routine of a program using the Scheduler. Each time processing of a set of transaction data has been completed by the disk routine, the input/output areas used by the data must be released, and control must be returned to another disk routine or to the main routine. This macro-instruction directs the Scheduler to develop all the coding required to handle these functions (Figure 25).



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1		IOCTL	LEAVE
0 2			

Figure 24. The IOCTL LEAVE Macro-Instruction

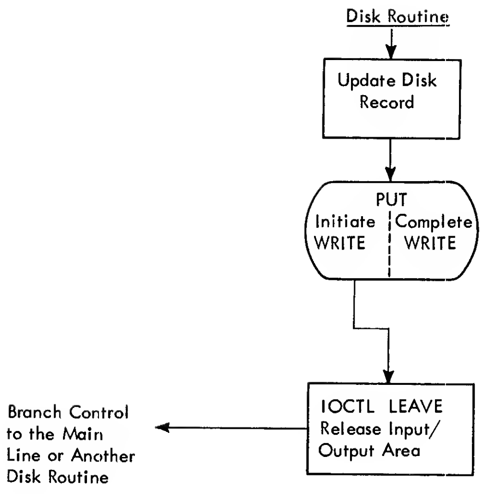

Disk Routine
Update Disk Record
PUT
Initiate WRITE
Complete WRITE
IOCTL LEAVE
Release Input/Output Area
Branch Control to the Main Line or Another Disk Routine

Figure 25. Use of the IOCTL LEAVE Macro-Instruction

What This Macro-Instruction Will Do: Each time the programmer uses the IOCTL LEAVE macro-instruction, the Scheduler will develop the coding required to:

1. check whether the data in the current segment of the Transaction Stacking Area is required by another disk routine, if any,
2. free the segment of the Transaction Stacking Area used by a completely processed transaction record,
3. send control to the subroutine defined in an IOCTL DEFSA macro-instruction, if this macro-instruction was used to define the transaction record to be released,
4. release the input/output area used by the GET macro-instruction immediately preceding the present IOCTL LEAVE macro-instruction (see the section describing the IOCTL FSEQP macro-instruction),
5. check whether another disk record is ready for processing in an input/output area,
6. branch control to the appropriate disk routine if another disk record is waiting to be processed,
7. check whether the Scheduler is saturated with requests, and
8. return control to the main routine if the Scheduler is not saturated with requests.

The DTF (Define The File) Statement

Purpose of the DTF Statement

The programmer who wishes to use the Random-Processing Scheduler must write a DTF (Define the File) statement for each disk file used by his program. This information consists of up to twelve entries. Each DTF statement describes the characteristics of the file for which it was written and indicates the methods to be used by the Scheduler in handling the file. Using the information supplied in the DTF statement, the Autocoder processor develops the File Table required for the proper handling of each file.

General Format of the DTF Statement

The first entry of a DTF statement is the DTF Header Line. It consists of the code, DTF, in the operation field followed by the name of the file in the operand field. All subsequent entries for that DTF statement have blank operation fields with the entry name placed in the label field. The entries following the header line may be listed in any order.

DTF entries without operands are not permitted.

All operands of the DTF entries may use address modification, where applicable, provided the operand consists of no more than 13 characters. Thus, LABEL+110 is a valid operand if LABEL consists of no more than nine characters. All symbolic operands of DTF entries, except those of input/output areas, may be indexed. (The number of characters used to designate the index register must be included in the count of 13.)

The DTF Entries

The following Scheduler DTF entries are described in the publication, *IBM 1410/7010 Operating System; Basic Input/Output Control System,* Form C28-0322. They may be used with the Scheduler as described in that publication, with two exceptions:

1. The INDEX DTF entry is required for random files.
2. The STORE operand of the ERRCHECK DTF entry must not be used when defining random files.

ERRCHECK
FILELIST
INDEX
IOAREAS
MODE
ORDER

The following entries have functions with the Scheduler other than those explained in the Basic IOCS publication. These other functions are described herein.

SYMUNIT
ERRADDR
ERROPTNS
FILEFORM

The following DTF entry is used only for the Random-Processing Scheduler:

ACTIVITY

ACTIVITY

The ACTIVITY entry is ordinarily used only with *input* files. (The only time the ACTIVITY entry is used in defining an *output* file is when that file will be referenced by a disk routine and will be the only type of random file currently open.)

The ACTIVITY entry has a single numeric operand, which the Scheduler uses to tailor the size of the scheduling routine to the user's requirements, so that the user will have maximum scheduling efficiency in a minimum of core storage.

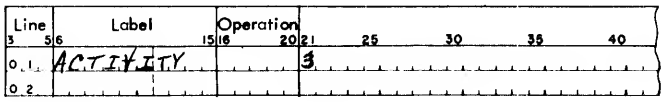
The value of the operand may be calculated by determining the processing time involved in executing the main routine and the disk routine through to the point where the first GET macro-instruction is encountered in the disk routine, and dividing this value into the average Seek time for the file concerned. The result of this division will be a value that can be used as the ACTIVITY operand for the file. This value represents the average number of requests for disk records that the disk routine may issue before data is made available to it from disk storage for processing.

A number approximating the optimum value for the ACTIVITY operand may be determined by adding one (1) to the number of input/output areas associated with the file, i.e., if two input/output areas are assigned to the file, the ACTIVITY operand would be three (3).

In choosing a value for the ACTIVITY operand, the following points should be considered:

1. The sum of the ACTIVITY operands for all files represents the maximum number of requests for disk records that the Scheduler will be able to accept before entering a waiting loop.
2. The sum of the ACTIVITY operands for all random input files opened must be at least 1. However, if the total activity defined for the Scheduler is only 1, there can be no overlap of input/output with processing.
3. When a random input file is only referred to following an IOCTL FSEQP macro-instruction, it does not require an ACTIVITY entry.
4. When input/output requests for two or more files are mutually exclusive, ACTIVITY need only be specified for one of the files.

The ACTIVITY entry is coded as shown in Figure 26.



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1	ACTIVITY		3
0 2			

Figure 26. The ACTIVITY DTF Entry

ERRADDR

The ERRADDR DTF entry is used in the same manner as described in the publication *IBM 1410/7010 Basic Input/Output Control System.* When using the ERRADDR DTF entry for random processing, the specified user-written routine must determine which option the Scheduler is to take in handling the error situation.

The user determines the nature of the error by analyzing the bit configuration in the error summary position of the IORW (18+X15), and specifies one of the following options to be taken in handling the error (Figure 27):

1. BXPA /RNR/ – repeat the GET or PUT macro-instruction for that record.
2. BXPA /RNP/ – process the record anyway.
3. BXPA /RNB/ – bypass the rest of the disk routine (cannot be used with dependent disk routines).

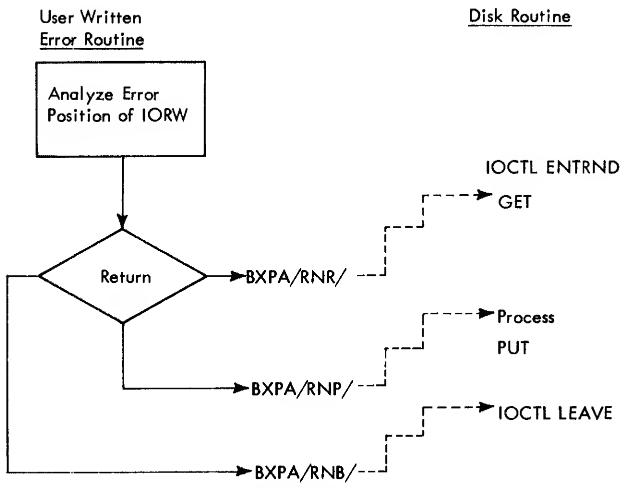


Figure 27. Directions to the Random-Processing Scheduler for Handling Error Situations, with Eventual Returns

NOTE: If it is desired to retry the GET with a new disk address, the user must place the new address into the two eight-position areas preceding the input/output area associated with the file. (See the "DA Statement" section for details of the input/output area.) The address of the label will be in the IORW with its low-order position at 14+X15. Figure 28 illustrates placement of addresses.

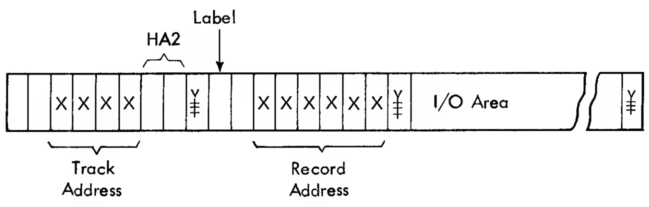


Figure 28. Address Placement

ERROPTNS

This entry is optional. It is used when the user desires to bypass an *independent* disk routine if the data record it is requesting cannot bc obtained without uncorrectable errors. If the user desires that the disk routine that requested the record be bypassed, then an ERROPTNS entry with the BYPASS operand must be included in the DTF statement for the file concerned (Figure 29). If an ERROPTNS entry is not included, the record with the uncorrectable error will be processed. This entry will not handle "no-record-found" conditions.

NOTE: This DTF entry must not be included for files that access records for dependent disk routines. If the cntry were used, a dependent disk routine might be bypassed because of an uncorrectable error, but the Scheduler would not bypass the additional disk routines that follow.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1	ERROPTNS		BYPASS
0 2			

Figure 29. The ERROPTNS DTF Entry

FILEFORM

This entry is required. The operand is fixed and must be RANDOM, as shown in Figure 30.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1	FILEFORM		RANDOM
0 2			

Figure 30. Thc FILEFORM DTF Entry

SYMUNIT (Symbolic Unit)

This entry is required. It differs from the SYMUNIT entry used in defining a file to the Basic IOCS, in that only one operand is used. This operand must be "1301." See Figure 31.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1	SYMUNIT		1301
0 2			

Figure 31. The SYMUNIT Entry

DA (Define Area) Statements Needed to Support the Random-Processing Scheduler

Some areas required by the Random-Processing Scheduler must be reserved by the programmer by means of an appropriate DA statement. (A discussion of DA statements may be found in the *Autocoder* publication.) All such areas must be terminated by a group mark with word mark immediately to the right of thc low-order position of the area.

The DA Statement for Input/Output Areas

Each program utilizing the Random-Processing Scheduler requircs at least one input/output area for each disk file used by the program. The input/output areas are used for storing and processing information obtained from disk storage. Each input/output area must be reserved by the programmer by an appropriate DA statement. They need not be contiguous in core storage.

As indicated in Figure 32, each input/output area must be preceded by two nine-position areas required by the Scheduler.

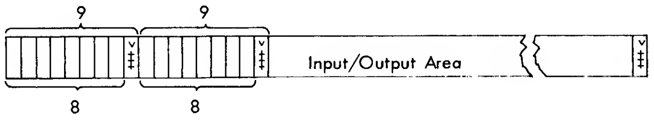


Figure 32. Format of an Input/Output Area

Figure 33 shows an example of the type of DA statement that must be written for an input/output area that will contain a 100-character record. LABEL must be the name assigned to this area in the DTF IOAREAS entry. Indexing and referencing to zero may be specified in the DA entry. The index register specified must be the one specified in the INDEX entry of the DTF statement for the file with which the input/output area is associated.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1		DA	1X8,G
0 2	LABEL	DA	1X8,G
0 3		DA	1X100,G
0 4			

Figure 33. DA Statement Defining an Input/Output Area

The DA Statement for the Transaction Stacking Area

The DA statement for a Transaction Stacking Area is of the form shown in Figure 34. The DCW entry is required by the Scheduler and must be coded as shown. The label of the DCW entry is the label used in the operands of the IOCTL STACK and IOCTL MVRSA macro-instructions.

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
0 1	ANYLABEL	DCW	00000
0 2		DA	NXM,G
0 3			

Figure 34. Coding a Transaction Stacking Area

The DA entry defines the Transaction Stacking Area.

N indicates the number of identical segments to be reserved.

M indicates the number of positions to be reserved for each segment. (This must include a position in each segment for a group mark with word mark or for a record mark.)

G specifies that a group mark with word mark is to be included at the end of the entire area reserved by this DA.

To simplify the problem of referencing fields within segments of the Stacking Area, the DA entry may specify indexing and referencing to zero. Together, these two entries enable the programmer to refer to the fields in a segment by name (label).

If the index register specified is the same as the index register defined by the IOCTL STACK macro-instruction for this Stacking Area, it will be set up and maintained by the Scheduler.

The number of segments, N, in the Transaction Stacking Area may be estimated from the following formula:

$$N = \frac{A}{B} + 1,$$

where:

A = The sum of the operands of the ACTIVITY parameters for all files contained within the disk routines for which the Transaction Stacking Area applies.

B = The number of disk routines for which this Transaction Stacking Area applies. Normally, any additional segments will waste core storage.

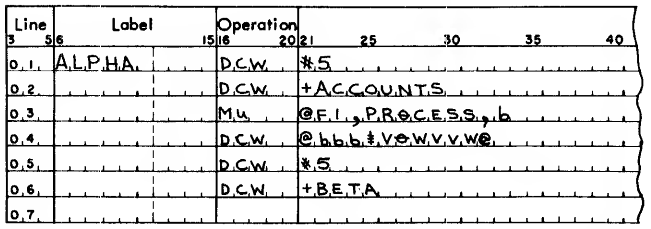
If variable-length transaction records are to be moved into the same Transaction Stacking Area, each segment must be large enough to hold the longest record.

Word marks may be specified for segment subfields at the user's discretion, e.g., if the programmer desires to move data to the Transaction Stacking Area. (See Format B of the IOCTL MVRSA macro-instruction.)

Input/Output Request Word for Random Processing

The Input/Output Request Words needed by the Scheduler are created automatically for the input/output areas defined in the DTF IOAREAS entry. If additional input/output areas are desired, the programmer must construct an IORW, define the additional input/output area and use the label of the first additional IORW he constructs as the operand in the FILELIST DTF entry for the file.

The IORW for the Random-Processing Scheduler is slightly different from that for the Basic IOCS. There are 40 positions in each IORW and these are grouped into six fields. Each field is described briefly. Figure 35 illustrates the coding.



Line 3 5	Label 6 15	Operation 16 20	21 25 30 35 40
01	ALPHA	DCW	*5
02		DCW	+ACCOUNTS
03		MU	@FI,PRØCESS,b
04		DCW	@bbb‡VØWVVW@
05		DCW	*5
06		DCW	+BETA
07			

Figure 35. Example: Coding of an Input/Output Request Word

Field 1 is a five-position field. It is not used by the Scheduler and may be either zeros or blanks. The low-order position of this field is the reference address that will be used in handling this IORW.

Field 2 is a five-position field. It contains the address of the List Origin of the File Table for that particular file.

Field 3 is a ten-position field. It will be the appropriate disk instruction that will be executed.

Field 4 is a ten-position field. It is a conglomerate field and the positions are described as follows:

Position 1
: is always a blank character with a word mark.

Position 2
: is always a blank character.

Position 3
: is an error indicator and is coded as a blank. When the Scheduler gives control to a user-written error routine, this position will contain a bit configuration corresponding to the error condition. (See the discussion on the ERRADDR DTF entry in this publication.)

Position 4
: is either a word separator character, if wrong-length-record checking is not desired for this file, or a group mark, if wrong-length-record checking is desired for this file. There must not be a word mark in this position.

Position 5
: is always a V.

Position 6
: is always the letter O (11-6 punch).

Position 7
: is a W, if the ERRADDR DTF entry is not being used for this file, or an O (11-6), if the ERRADDR DTF entry is being used for this file.

Position 8
: is always a V.

Position 9
: is always a V.

Position 10
: is a V, if write disk check is not desired for this file, or a W, if write disk check is desired.

Field 5 is a five-position field and is used by the Scheduler.

Field 6 is a five-position field and is the permanent link field. For random processing, it functions just as Field 1 functions for the Basic IOCS. That is, it must contain either the address of the low-order position of Field 1 in the next IORW, or zeros if this is the last IORW.

The input/output area to accompany the user-constructed IORW is defined as shown in the preceding section. The label of the input/output area must be used as the B-address of the instruction coded in Field 3 of the IORW.

Index

Reader's Comments

IBM 1410/7010 Operating System (1410-PR-155)
Random-Processing Scheduler-1410-IO-967

Form C28-0323-1

From

Name ____

Address ____

Your comments regarding the completeness, clarity, and accuracy of this publication will help us improve future editions. Please check the appropriate items below, add your comments, and mail.

	YES	NO
Does this publication meet the needs of you and your staff?	____	____
Is this publication clearly written?	____	____
Is the material properly arranged?	____	____

If the answer to any of these questions is "NO," be sure to elaborate.

How can we improve this publication? Please answer below.

☐ Suggested Addition (Page , Timing Chart, Drawing, Procedure, etc.)

☐ Suggested Deletion (Page)

☐ Error (Page)

COMMENTS:

No Postage Necessary if Mailed in U.S.A.

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N.Y.

ATTN : PROGRAMMING SYSTEMS PUBLICATIONS
DEPARTMENT D91

CUT ALONG LINE

FOLD FOLD

STAPLE STAPLE

C28-0323-1

Printed in U.S.A. C28-0323-1

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York